Table of Contents

Verification of satellite rendezvous problem via SReachSet

This example will demonstrate the use of SReachTools in verification of stochastic continuous-state discrete-time linear time-invariant (LTI) systems.

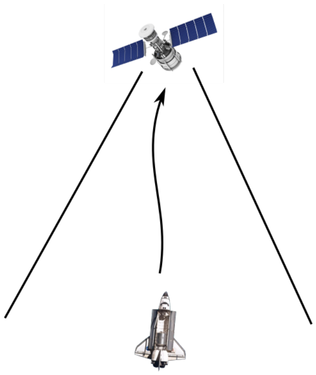
Specifically, we will discuss how SReachTools can use Fourier transforms (Genz's algorithm and MATLAB's patternsearch), convex chance constraints, and Lagrangian methods to construct underapproximative stochastic reach sets.

Our approaches is grid-free and recursion-free resulting in highly scalable solutions, especially for Gaussian-perturbed LTI systems.


```
% Prescript running
close all;
% clc;
clearvars;
srtinit
```

Problem formulation: Spacecraft motion via CWH dynamics

We consider both the spacecrafts, referred to as the deputy spacecraft and the chief spacecraft, to be in the same circular orbit. In this example, we will consider the problem of verification for the spacecraft rendezvous problem, i.e., identify all the initial states from which the deputy can can rendezvous with the chief while staying within the line-of-sight cone with a likelihood above a user-specified threshold.



Dynamics model for the deputy relative to the chief spacecraft

The relative planar dynamics of the deputy with respect to the chief are described by the [Clohessy-Wiltshire-Hill (CWH) equations,](#)

$$\ddot{x} - 3\omega x - 2\omega \dot{y} = \frac{F_x}{m_d}$$

$$\ddot{y} + 2\omega \dot{x} = \frac{F_y}{m_d}$$

where the position of the deputy relative to the chief is $x, y \in \mathbf{R}$, $\omega = \sqrt{\frac{\mu}{R_0^3}}$ is the orbital frequency, μ is the gravitational constant, and R_0 is the orbital radius of the chief spacecraft. We define the state as $\overline{x} = [x\ y\ \dot{x}\ \dot{y}]^\top \in \mathbf{R}^4$ which is the position and velocity of the deputy relative to the chief along x- and y- axes, and the input as $\overline{u} = [F_x\ F_y]^\top \in \mathcal{U} \subset \mathbf{R}^2$.

We will discretize the CWH dynamics in time, via zero-order hold, to obtain the discrete-time linear time-invariant system and add a Gaussian disturbance to account for the modeling uncertainties and the disturbance forces,

$$\overline{x}_{k+1} = A\overline{x}_k + B\overline{u}_k + \overline{w}_k$$

with $\overline{w}_k \in \mathbf{R}^4$ as an IID Gaussian zero-mean random process with a known covariance matrix $\Sigma_{\overline{w}}$.

System definition

```
umax = 0.1;
mean_disturbance = zeros(4,1);
covariance_disturbance = diag([1e-4, 1e-4, 5e-8, 5e-8]);
% Define the CWH (planar) dynamics of the deputy spacecraft relative
 to the
% chief spacecraft as a LtiSystem object
sys = getCwhLtiSystem(4, Polyhedron('lb', -umax*ones(2,1), ...
                                    'ub',  umax*ones(2,1)), ...
```

```
      RandomVector('Gaussian',
 mean_disturbance,covariance_disturbance));
```

Methods to run

```
ft_run = 0;
cc_open_run = 1;
lagunder_run = 0;
```

Target tube construction --- reach-avoid specification

```
time_horizon = 5;          % Stay within a line of sight cone for 4
 time steps and
                         % reach the target at t=5% Safe Set --- LoS
 cone
% Safe set definition --- LoS cone |x|<=y and y\in[0,ymax] and |vx|
<=vxmax and
% |vy|<=vymax
ymax = 2;
vxmax = 0.5;
vymax = 0.5;
A_safe_set = [1, 1, 0, 0;
             -1, 1, 0, 0;
              0, -1, 0, 0;
              0, 0, 1,0;
              0, 0,-1,0;
              0, 0, 0,1;
              0, 0, 0,-1];
b_safe_set = [0;
              0;
              ymax;
              vxmax;
              vxmax;
              vymax;
              vymax];
safe_set = Polyhedron(A_safe_set, b_safe_set);
% Target set --- Box [-0.1,0.1]x[-0.1,0]x[-0.01,0.01]x[-0.01,0.01]
target_set = Polyhedron('lb', [-0.1; -0.1; -0.01; -0.01], ...
                        'ub', [0.1; 0; 0.01; 0.01]);
target_tube = Tube('reach-avoid',safe_set, target_set, time_horizon);
slice_at_vx_vy = zeros(2,1);
```

Preparation for set computation

```
prob_thresh = 0.8;

n_dir_vecs = 10;
theta_vec = linspace(0, 2*pi, n_dir_vecs);
set_of_dir_vecs_ft = [cos(theta_vec);
```

```
                      sin(theta_vec);
                      zeros(2,n_dir_vecs)];
n_dir_vecs = 40;
theta_vec = linspace(0, 2*pi, n_dir_vecs);
set_of_dir_vecs_cc_open = [cos(theta_vec);
                           sin(theta_vec);
                           zeros(2,n_dir_vecs)];
init_safe_set_affine = Polyhedron('He',[zeros(2,2) eye(2,2)
 slice_at_vx_vy]);
```

CC (Linear program approach)

```
if cc_open_run
    options = SReachSetOptions('term', 'chance-open', ...
        'set_of_dir_vecs', set_of_dir_vecs_cc_open, ...
        'init_safe_set_affine', init_safe_set_affine, ...
        'verbose', 0);
    timer_cc_open = tic;
    [polytope_cc_open, extra_info] = SReachSet('term','chance-open',
 sys, ...
        prob_thresh, target_tube, options);
    elapsed_time_cc_open = toc(timer_cc_open);
end
```

Fourier transform (Genz's algorithm and MATLAB's patternsearch)

```
if ft_run
    options = SReachSetOptions('term', 'genzps-open', ...
        'set_of_dir_vecs', set_of_dir_vecs_ft, ...
        'init_safe_set_affine', init_safe_set_affine, 'verbose', 1);
    timer_ft = tic;
    polytope_ft = SReachSet('term','genzps-open', sys,
 prob_thresh, ...
        target_tube, options);
    elapsed_time_ft = toc(timer_ft);
end
```

Lagrangian approach

```
if lagunder_run
    options = SReachSetOptions('term', 'lag-
under', 'bound_set_method', ...
        'ellipsoid');

    timer_lagunder = tic;
    polytope_lagunder = SReachSet('term', 'lag-under', sys,
 prob_thresh, ...
        target_tube, options);
    elapsed_time_lagunder = toc(timer_lagunder);
end
```

Preparation for Monte-Carlo simulations of the optimal controllers

Monte-Carlo simulation parameters

```
n_mcarlo_sims = 1e5;
n_sims_to_plot = 5;
```

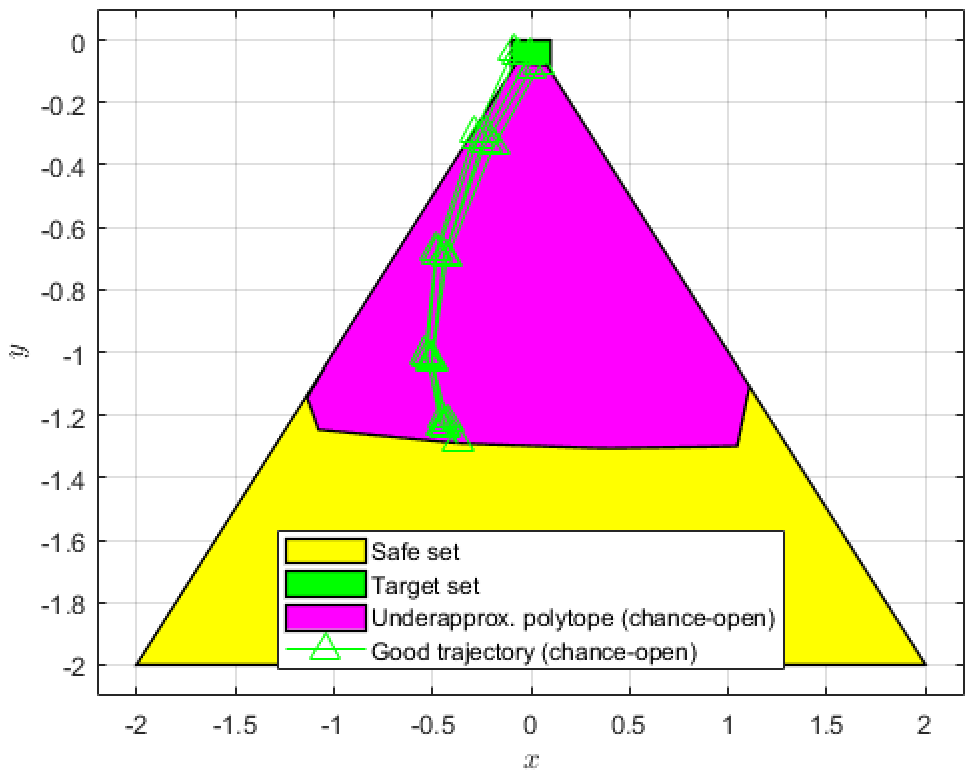
Plotting and Monte-Carlo simulation-based validation

```
figure(1);
clf
box on;
hold on;
plot(safe_set.slice([3,4], slice_at_vx_vy), 'color', 'y');
plot(target_set.slice([3,4], slice_at_vx_vy), 'color', 'g');
legend_cell = {'Safe set','Target set'};
if exist('polytope_cc_open','var') && ~isEmptySet(polytope_cc_open)

 plot(Polyhedron('V',polytope_cc_open.V(:,1:2)), 'color','m','alpha',1);
    legend_cell{end+1} = 'Underapprox. polytope (chance-open)';
else
    polytope_cc_open = Polyhedron();
    elapsed_time_cc_open = NaN;
end
if exist('polytope_ft','var') && ~isEmptySet(polytope_ft)
    plot(Polyhedron('V',polytope_ft.V(:,1:2)), 'color','b','alpha',1);
    legend_cell{end+1} = 'Underapprox. polytope (genzps-open)';
else
    polytope_ft = Polyhedron();
    elapsed_time_ft = NaN;
end
if exist('polytope_lagunder','var') && ~isEmptySet(polytope_lagunder)
    %plot(polytope_lagunder.slice([3,4], slice_at_vx_vy),
 'color','r','alpha',1);
    plot(Polyhedron('V',polytope_ft.V(:,1:2)), 'color','r','alpha',1);
    legend_cell{end+1} = 'Underapprox. polytope (lag-under)';
else
    polytope_lagunder = Polyhedron();
    elapsed_time_lagunder = NaN;
end
direction_index_to_plot = 30;
if ~isEmptySet(polytope_cc_open)
    init_state =
 extra_info(2).vertices_underapprox_polytope(:,direction_index_to_plot);
    input_vec =
 extra_info(2).opt_input_vec_at_vertices(:,direction_index_to_plot);
    opt_reach_avoid =
 extra_info(2).opt_reach_prob_i(direction_index_to_plot);
```

```
    concat_state_realization = generateMonteCarloSims(...
            n_mcarlo_sims, ...
            sys, ...
            init_state, ...
            time_horizon, ...
            input_vec);

    % Check if the location is within the target_set or not
    mcarlo_result = target_tube.contains(concat_state_realization);
    [legend_cell] = plotMonteCarlo(' (chance-open)',
 mcarlo_result, ...
        concat_state_realization, n_mcarlo_sims, n_sims_to_plot, ...
        sys.state_dim, init_state, legend_cell);
end
legend(legend_cell, 'Location','South');
xlabel('$x$','interpreter','latex');
ylabel('$y$','interpreter','latex');
fprintf('Expected probability: %1.3f, Simulated probability: %1.3f
\n',...
    opt_reach_avoid, sum(mcarlo_result)/n_mcarlo_sims);
```



Reporting solution times

```
if any(isnan([elapsed_time_ft, elapsed_time_cc_open,
 elapsed_time_lagunder]))
```

```
    disp('Skipped items would show up as NaN');
end
fprintf(['Elapsed time: (genzps-open) %1.3f | (chance-open) %1.3f',...
    ' | (lag-under) %1.3f seconds\n'], ...
    elapsed_time_ft, elapsed_time_cc_open, elapsed_time_lagunder);
```

Skipped items would show up as NaN
Elapsed time: (genzps-open) NaN | (chance-open) 23.156 | (lag-under) NaN seconds

Helper functions

Plotting function

```
function [legend_cell] = plotMonteCarlo(method_str, mcarlo_result, ...
    concat_state_realization, n_mcarlo_sims, n_sims_to_plot,
 state_dim, ...
    initial_state, legend_cell)
% Plots a selection of Monte-Carlo simulations on top of the plot

    green_legend_updated = 0;
    red_legend_updated = 0;
    traj_indices = floor(n_mcarlo_sims*rand(1,n_sims_to_plot));
    for realization_index = traj_indices
        % Check if the trajectory satisfies the reach-avoid objective
        if mcarlo_result(realization_index)
            % Assign green triangle as the marker
            markerString = 'g^-';
        else
            % Assign red asterisk as the marker
            markerString = 'r*-';
        end
        % Create [x(t_1) x(t_2)... x(t_N)]
        reshaped_X_vector = reshape(...
            concat_state_realization(:,realization_index), state_dim,
[]);
        % This realization is to be plotted
        h = plot([initial_state(1), reshaped_X_vector(1,:)], ...
                 [initial_state(2), reshaped_X_vector(2,:)], ...
                 markerString, 'MarkerSize',10);
        % Update the legends if the first else, disable
        if strcmp(markerString,'g^-')
            if green_legend_updated
                h.Annotation.LegendInformation.IconDisplayStyle
 = 'off';
            else
                green_legend_updated = 1;
                legend_cell{end+1} = strcat('Good trajectory ',
 method_str);
            end
        elseif strcmp(markerString,'r*-')
            if red_legend_updated
                h.Annotation.LegendInformation.IconDisplayStyle
 = 'off';
```

```
            else
                red_legend_updated = 1;
                legend_cell{end+1} = strcat('Bad trajectory ',
 method_str);
            end
        end
    end
end
```

Expected probability: 0.800, Simulated probability: 0.810

Published with MATLAB® R2017a